OPTIMALIZACE CESTY POMOCÍ ALGORITMU RRT V PROSTŘEDÍ MATLAB

Petr Šoustek, Petr Krček, Jiří Dvořák, Daniel Zuth, Radomil Matoušek

Department of Applied Computer Science
Faculty of Mechanical Engineering, Brno University of Technology

Abstrakt

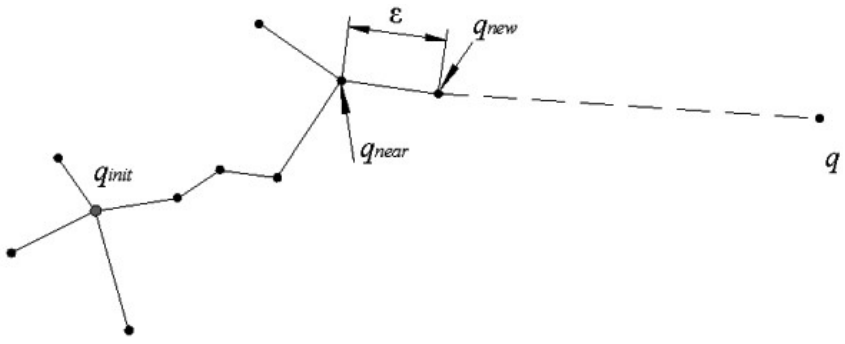
Problematika návrhu optimální cesty je z hlediska mobilních prostředků (robotů a manipulátorů) významná. Přes existenci poměrně sofistikovaných algoritmů založených například na principu potenciálových polí, či Voronoiových diagramů, existuje metoda tzv. "náhodně rostoucích stromů" (RRT - Rapidly-exploring Random Tree), která je schopna úlohu optimální cesty rovněž uspokojivě řešit. Metoda RRT [Steve LaValle, 1998] je v podstatě datová struktura i algoritmus, který umožňuje efektivně prohledávat vícedimenzionální nekonvexní prostor. V rámci prostředí Matlab existuje několik implementací RRT algoritmu, tento příspěvek představuje jednu z nich.

1 Princip algoritmu RRT

Důležitým rysem inteligentních systémů je schopnost vytvořit si vnitřní model prostředí a pracovat s ním. Je-li zadán počáteční a cílový stav, má systém za úkol nalézt takovou posloupnost akcí, aby se jejich provedením dostal z počátečního stavu do cílového. Tato posloupnost akcí se nazývá plán. Stavový prostor můžeme reprezentovat orientovaným grafem, ve kterém uzly představují jednotlivé stavy a orientované hrany přechod mezi stavy. Úlohu plánování pohybu tedy můžeme formulovat jako hledání cesty v grafu stavového prostoru. K nalezení vhodné cesty robota je třeba použít některé z plánovacích metod. V následujícím přehledu je popsáno několik nejčastěji používaných metod pro plánování ve dvourozměrném prostoru:

- Metody rozkladu do buněk (Cell decomposition).
- Mapy cest (Roadmaps).
- Potenciálová pole (Potential fields).

Metoda RRT (A Rapidly exploring Random Tree) byla poprvé představena Stevem LaValle v roce 1998. Tuto metodu můžeme zařadit do skupiny pravděpodobnostních plánovačů map cest. Pojem RRT lze přeložit jako rychle rostoucí náhodné stromy a v základu chápat jako specifickou datovou strukturu a algoritmus určený pro efektivní prohledávání nekonvexního n-dimenzionálního metrického prostoru. Metoda RRT je využitelná jak pro holonomní, tak neholonomní mobilní roboty, pohybující se ve statickém i dynamickém prostředí. V průběhu řešení problému (plánování cesty) se RRT datová struktura postupně rozrůstá a vytváří nové uzly stromové struktury ve směru náhodně vybraných bodů. RRT začíná ve startovní konfiguraci q_{init} a během expanze se snaží nalézt cílovou konfiguraci q_{goal}. V každém kroku ε se rozpíná do náhodné konfigurace q_{rand}.



Obrázek 1: Princip růstu RRT stromu.

V průběhu hledání jsou vrcholy stromu tvořeny takovým způsobem, že všechny uzly náleží C_{free} (volný konfigurační prostor) Z tohoto důvodu musí existovat možnost testovat, zda určitý stav leží v C_{obst} . Stavy v C_{obst} mohou mít v závislosti na aplikaci různý význam. Nejčastěji C_{obst} představuje konfiguraci, kdy je robot v kolizi s překážkou. Podrobný popis algoritmu a jeho variant je možné nalézt v [LaValle].

2 Matlab implementace RRT

Existuje několik implementaci metody RRT pro prostředí Matlab. Jako zdroj inspirace může sloužit odkaz [3, 4]. Naše varianta (implementace a interaktivní demo aplikace) vycházela z požadavku jednoduchého návrhu konfiguračního 2D prostoru a jednoduché RRT implementace (daná implementace je získatelná prostřednictvím emailu autorů). Překážky v konfiguračním prostoru jsou definovány formou matic znaků. Funkce `loadMatrix` zjistí rozměr matice a počet překážek. Překážky jsou vykresleny v souřadném systému, kdy *x*-ová souřadnice jednotlivé překážky odpovídá *m*-tému sloupci matice a *y*-ová souřadnice *n*-tému řádku.

```
%matice znaků pro generování bludiště
handles.m1 = ['0--0-00-----0-00000-'
              '00-0----000-0---0-0-'
              '---0000-------0-----'
              '0-----00-0-00000-0--'
              '--0-0-0--0-------00-'
              '00000-00-0--0-0-----'
              '---------00-000000--'
              '-0-00-0--0--0---0---'
              '---0--00-0-00-0-0-0-'
              '0--00-0-----0-0-000-'
              '0-----00-00-0-0-0---'
              '00000-0--0----------'
              '---------00-000000--'
              '0-00000000-------00-'
              '0-0--------00-0-----'
              '0-0--0-0-0--00000-0-'
              '0-00-000000---------'
              '--0--0-0--0-00-0-0--'];
```

```
function loadMatrix(m)
    cla;
    SizeMaze = size(m);
    NumLine = SizeMaze(1);
    NumColumn = SizeMaze(2);
    world.NumObstacles = 0;
    th = [(pi/4):(pi/2):(pi/4)+2*pi];

    hold on
    fill(0*sin(th) + 1, 0*cos(th) + 1,'r');
    fill(0*sin(th) + 99, 0*cos(th) + 99,'r');

    for y= 1:NumLine
        for x = 1:NumColumn
            if (m(y,x)=='0')

                px = (3.55*sin(th) + x*5);
                py = 100 - (3.55*cos(th) + y*5);

                fill(px,py,'b');
            end
        end
    end
```

Obrázek 2: Princip definování konfiguračního prostoru a příslušné obslužné funkce.

Počáteční a cílová pozice je v naší aplikaci zadávána poklikem do plochy bludiště pomocí funkce Matlabu `ginput`.V případě že je start nebo cíl zadán do oblasti obsazené překážkou, není tento bod akceptován a je potřeba jej zadat znovu. Případná kolize zadané konfigurace se ověřuje funkcí `collisionObstacle` pro každou překážku bludiště zvlášť.

```
%funkce ověřuje zda zadaná konfigurace
nekoliduje s překážkou
function result = collisionObstacle(m,
scx, scy)
result = 0;
 SizeMaze = size(m);
    NumLine = SizeMaze(1);
    NumColumn = SizeMaze(2);
    world.NumObstacles = 0;
    th = [(pi/4):(pi/2):(pi/4)+2*pi];

    hold on
    fill(0*sin(th) + 1, 0*cos(th) +
1,'r');
    fill(0*sin(th) + 99, 0*cos(th) +
99,'r');
```

```
for y= 1:NumLine
    for x = 1:NumColumn
        if (m(y,x)=='0')

            px = (3.55*sin(th) + x*5);
            py = 100 - (3.55*cos(th) + y*5);

            if (((scx>x*5-2.5) && (scx<x*5+2.5))
...
               && (((100-scy)>y*5-2.5) && ((100-
scy)<y*5+2.5)))
                fill(px,py,'r');
                result = 1;
            end
        end
    end
end
```

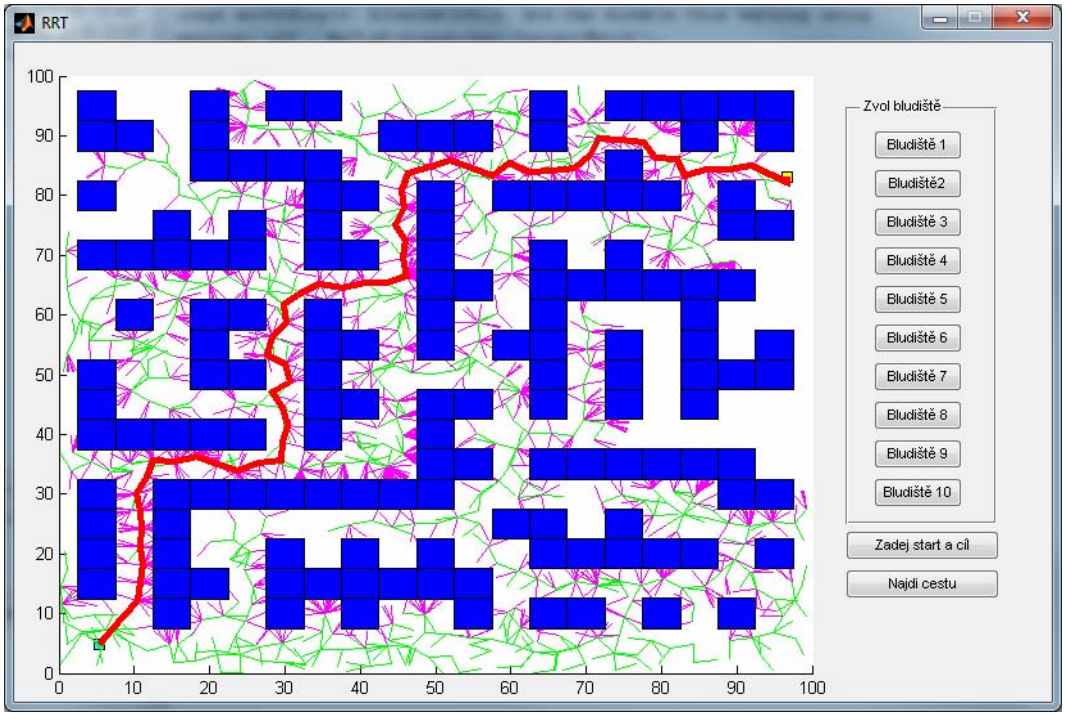
Obrázek 3: Detekce kolizí

Funkcí `createWorld` je vytvářena datová struktura světa s překážkami.

```
function world = createWorld(m,NumObstacles, NEcorner, SWcorner);

  if (NEcorner(1) <= SWcorner(1)) | (NEcorner(2) <= SWcorner(2)),
      disp('Not valid corner specifications!')
      world=[];

  % create world data structure
  else
    world.NEcorner = NEcorner;
    world.SWcorner = SWcorner;

    SizeMaze = size(m);
    NumLine = SizeMaze(1);
    NumColumn = SizeMaze(2);
    world.NumObstacles = 0;
    for y= 1:NumLine
        for x = 1:NumColumn
            if (m(y,x)=='0')

               world.radius(world.NumObstacles + 1)= 3.55;
               world.cn(world.NumObstacles + 1)= x*5;
               world.ce(world.NumObstacles + 1)= 100-(y*5);
               world.NumObstacles = world.NumObstacles + 1;

            end
        end
    end
   end
```

Obrázek 4: "Vytvoření světa" - datová struktura bludiště.

V dalším kroku je vygenerován náhodný bod, funkce `collision` ověří, jestli nekoliduje s některou z překážek. Kontrola kolize se provede tak, že se z rodičovského uzlu (předchozího přidaného bodu) ve směru náhodného bodu vygeneruje pět kontrolních bodů. Pro každý z nich se pak porovnává vzdálenost jeho *x*-ové a *y*-ové souřadnice se souřadnicemi každé překážky. Pokud je vzdálenost souřadnic menší nebo rovna zadanému poloměru překážky, je tento stav vyhodnocen jako kolize. Vyhoví-li podmínce všechny kontrolní body, můžeme strom o náhodný bod rozšířit, protože vzniklý segment cesty nebude kolidovat s žádnou z překážek. Pro započítání rozměru robota je v kontrolní podmínce kolize počítáno s větším rozměrem překážky, resp. oblastí překážek.



Obrázek 5: Příklad implementace RRT a nalezené řešení cesty.

```
function collision_flag = collision(node, parent, world);

collision_flag = 0;

if ((node(1)>world.NEcorner(1))...
    | (node(1)<world.SWcorner(1))...
    | (node(2)>world.NEcorner(2))...
    | (node(2)<world.SWcorner(2)))
  collision_flag = 1;
else

    plot([node(1),parent(1)],[node(2),parent(2)],'g');
    for sigma = 0:.2:1,
    p = sigma*node(1:2) + (1-sigma)*parent(1:2);
      % check each obstacle
      for i=1:world.NumObstacles,
        if (norm([p(1);p(2)]-[world.cn(i); world.ce(i)])<=4),
            collision_flag = 1;
            plot([node(1),parent(1)],[node(2),parent(2)],'m');
            break;
        end
      end
    end
end
```

Obrázek 6: Funkce pro zjištění kolizí.

Rozrůstání stromové struktury zajišťuje funkce `extendTree`. RRT strom se rozrůstá ve směru náhodně generovaných bodů o pevně danou délku segmentu kroku. Průběžně se kontroluje, zda je možné spojit nově přidaný bod s bodem cílovým a propočítává se minimální délka cesty `findMinimumPath`. Nalezenou cestu vykresluje funkce `plotWorld`. Stromová struktura je vykreslena zeleně a segmenty stromu vyhodnocené jako kolizní mají fialovou barvu. Výsledná cesta je zobrazena červeně.

3 Závěr

Prezentovaná aplikace RRT navržená ke studijním účelům je implementována čistě v prostředí Matlab. Této implementaci odpovídá rychlost generování RRT stromu a prohledávání stromové struktury. Kritickým místem algoritmu je pomocná funkce pro zjištění kolizí (Obr. 6). Celkové časy řešení pro různé hustoty konstantního rozměru bludiště (100x100) se pohybují v rozsahu do 30s. Na daném návrhu hodláme v další práci ověřit vliv velikosti kroku na rychlost konvergence vzhledem k hustotě překážek.

Poděkování

Práce vznikla jako součást řešení standardního SPP FSI BD13001025.

References

[1] Svidenská, A.: Bakalářská práce - RRT Plánování pohybu robota aplikovaná ve 2D prostoru. VUT v Brně, Brno, ČR, 2007.
[2] LaValle, S.M.: Planning Algorithms, 2006, www: http://msl.cs.uiuc.edu/~lavalle/
[3] Berard, R.: pathRRT, 2007, Last update 1. 1. 2010 http://www.et.byu.edu/groups/eceuavweb/labs/guidance_labs/PathPlanning/index.html
[4] Paul, G., Clifton, M.: Function RRT, Last update 16. 9. 2008 http://www.mathworks.com/matlabcentral/fx_files/21443/1/content/RRT20080916/html/RRT.htm

Adresa:
ÚAI, Fakulta strojního inženýrství, VUT v Brně, Technická 2, 616 69, Brno, ČR
email: {krcek@fme.vutbr.cz, zuth@fme.vutbr.cz, dvorak@fme.vutbr.cz, matousek@fme.vutbr.cz}